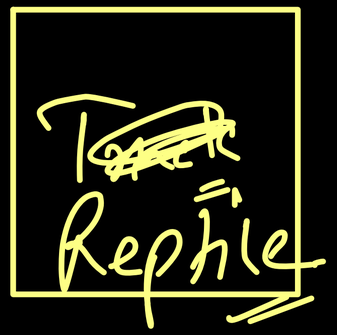
Inheritance

1. Similar to Inheritance in Life.
2. Child inherits attributes and behaviours from Parent.
3. Inheritance is a way to create a Class from existing Class.
4. The Derived / Child / Sub class inherits some attributes and behaviours from Base / Parent / Super Class, and may have more specialised attributes and behaviours.

When do we use Inheritance?

1. After grasping the definition, the pivotal question arises: "When do we use inheritance?”
2. Use inheritance whenever an IS-A relationship is identified between objects.
3. Inheritance is applied where objects exhibit a hierarchical relationship, signifying a specialized-generalized connection.
4. Do not Reinvent the Wheel.

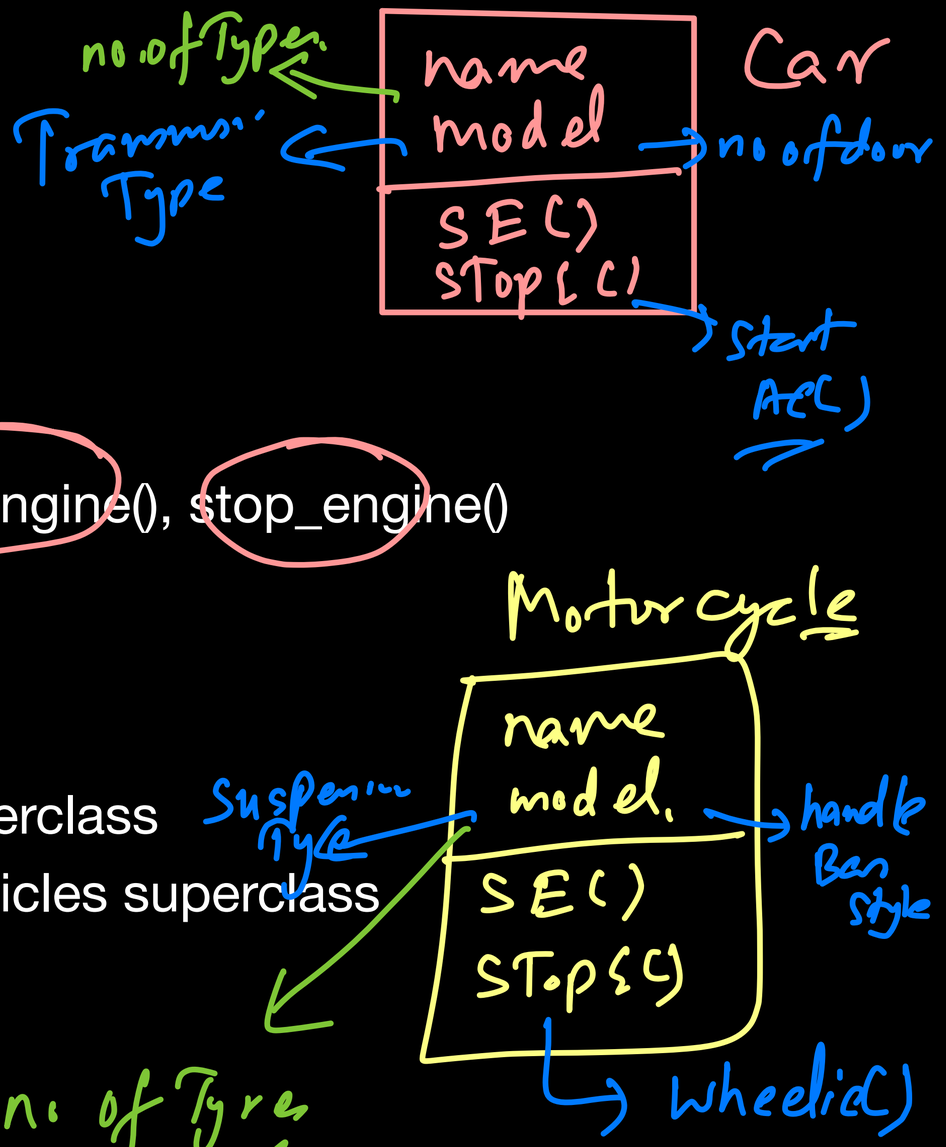
General “is-a” Examples

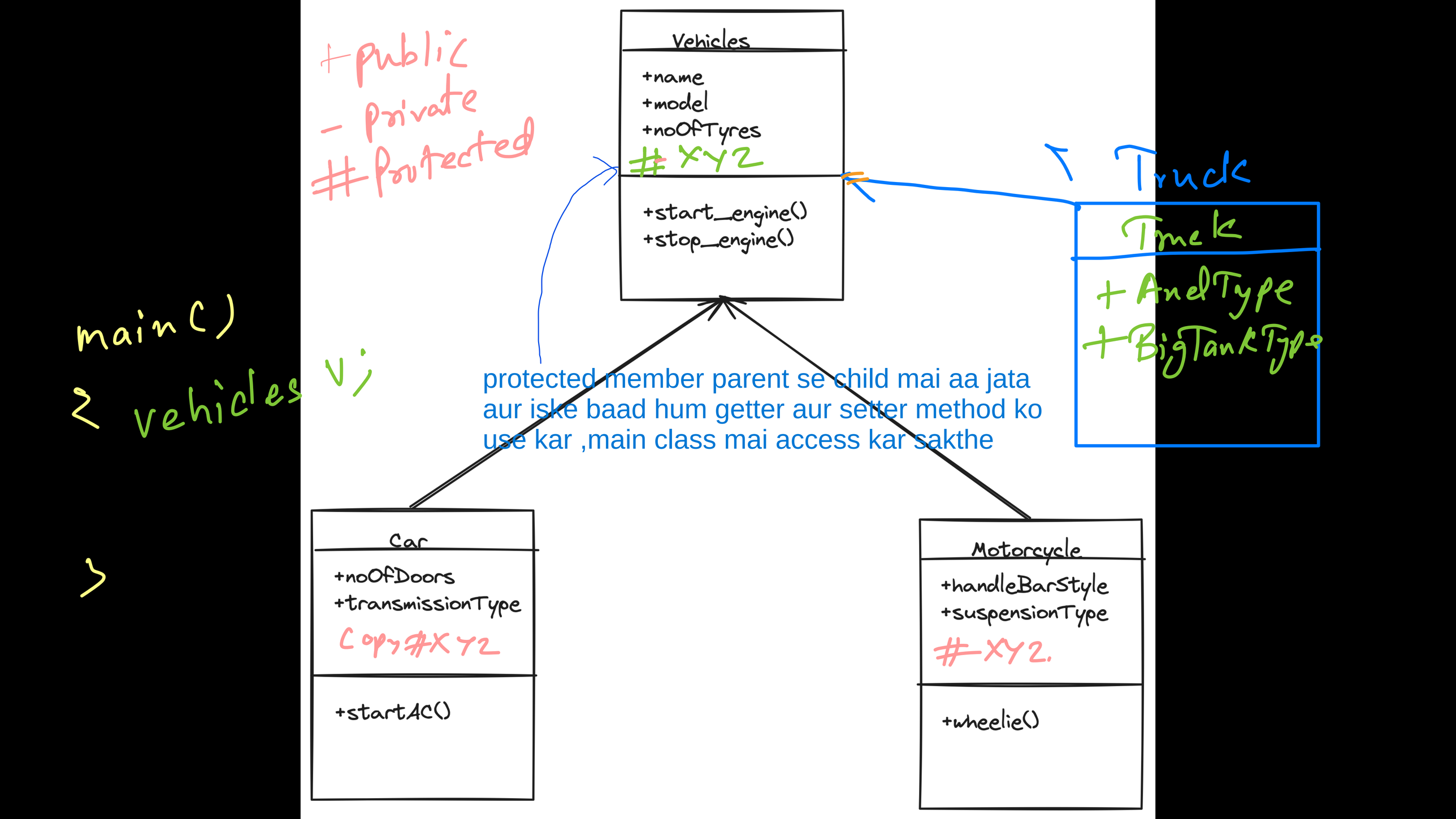


1. **Animal Hierarchy:**
   - Base Class: Animal
   - Derived Classes: Mammal, Reptile, Bird
   - Explanation: Mammals, reptiles, and birds are all types of animals, forming an "IS-A" relationship.
2. **Vehicle Classification:**
   - Base Class: Vehicle
   - Derived Classes: Car, Motorcycle, Truck
   - Explanation: Cars, motorcycles, and trucks are specific types of vehicles, demonstrating an "IS-A" relationship.
3. **Employee Categorization:**
   - Base Class: Employee
   - Derived Classes: Manager, Developer, HR Specialist
   - Explanation: Managers, developers, and HR specialists are employees with specific roles, establishing an "IS-A" relationship.
4. **Geometry Classes:**
   - Base Class: Shape
   - Derived Classes: Circle, Square, Triangle
   - Explanation: Circles, squares, and triangles are different shapes, forming an "IS-A" relationship within the context of geometry.
5. **Electronic Devices:**
   - Base Class: Device
   - Derived Classes: Smartphone, Laptop, Tablet
   - Explanation: Smartphones, laptops, and tablets are all electronic devices, creating an "IS-A" relationship.

Classic Example

- Program to represent different types of vehicles
- Represent entities like cars, motorcycles, etc.
- For cars, attributes like name, model; methods like start_engine(), stop_engine()
- Similar attributes and methods for the Motorcycle class
- Introduction of inheritance
- Create a superclass "Vehicles"
- Define general attributes and methods in the Vehicles superclass
- Cars and Motorcycles, as subclasses, inherit from the Vehicles superclass




+public
- private
#Protected
Vehicles
+name
+model
+noOfTyres
#XYZ
+start_engine()
+stop_engine()
Truck
Truck
+AxelType
+BigTankType
protected member parent se child mai aa jata aur iske baad hum getter aur setter method ko use kar ,main class mai access kar sakthe
Car
+noOfDoors
+transmissionType
Copy #XYZ
+startAC()
Motorcycle
+handleBarStyle
+suspensionType
#XYZ.
+wheelie()
main()
{
Vehicles V;
}

Mode of Inheritance

1. What scope of attributes of parents will be inherited to child class and with what scope?

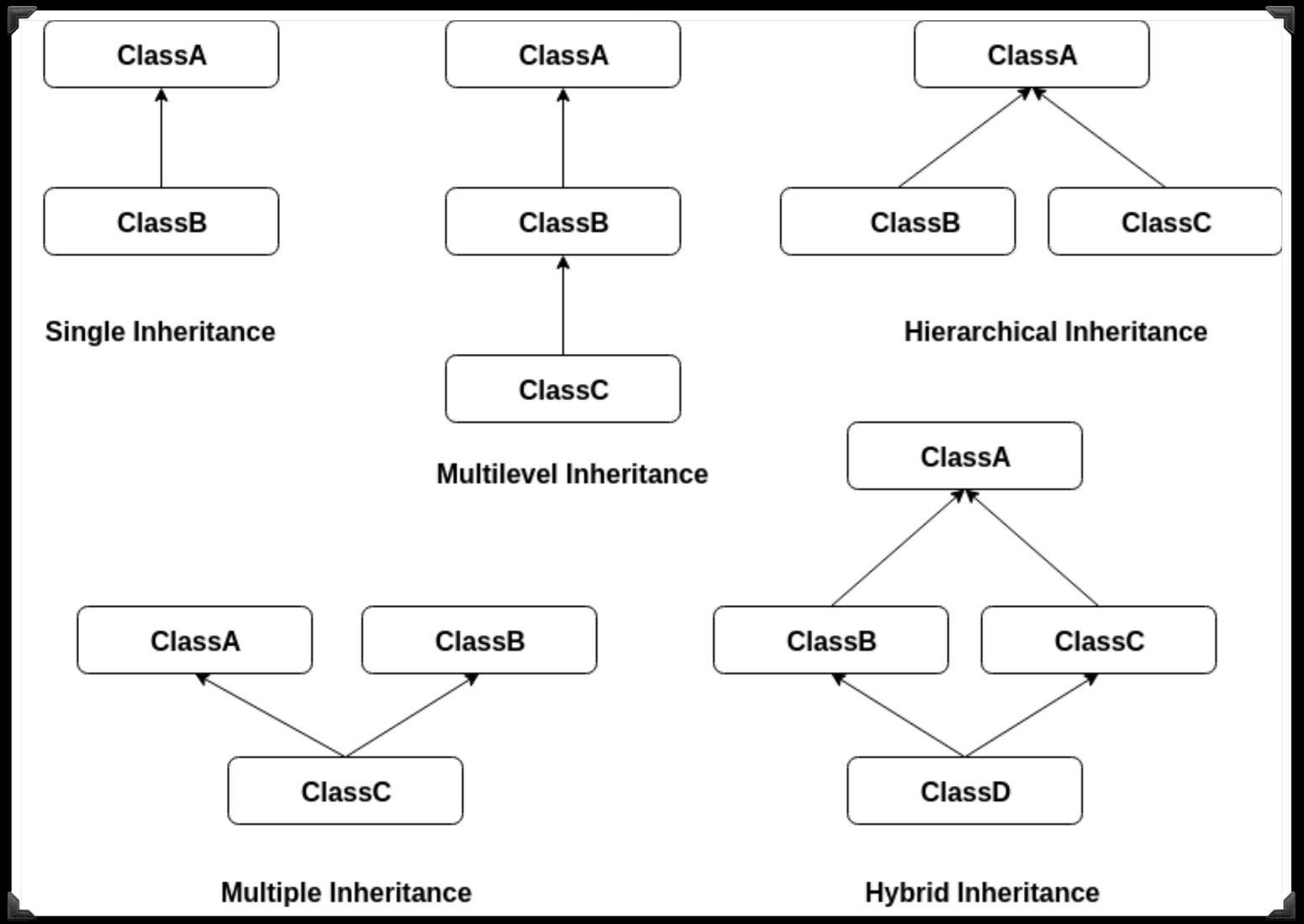
	Derived Class	Derived Class	Derived Class
Base Class	**Private Mode**	**Protected Mode**	**Public Mode**
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Private	Protected	Protected
Public	Private	Protected	Public

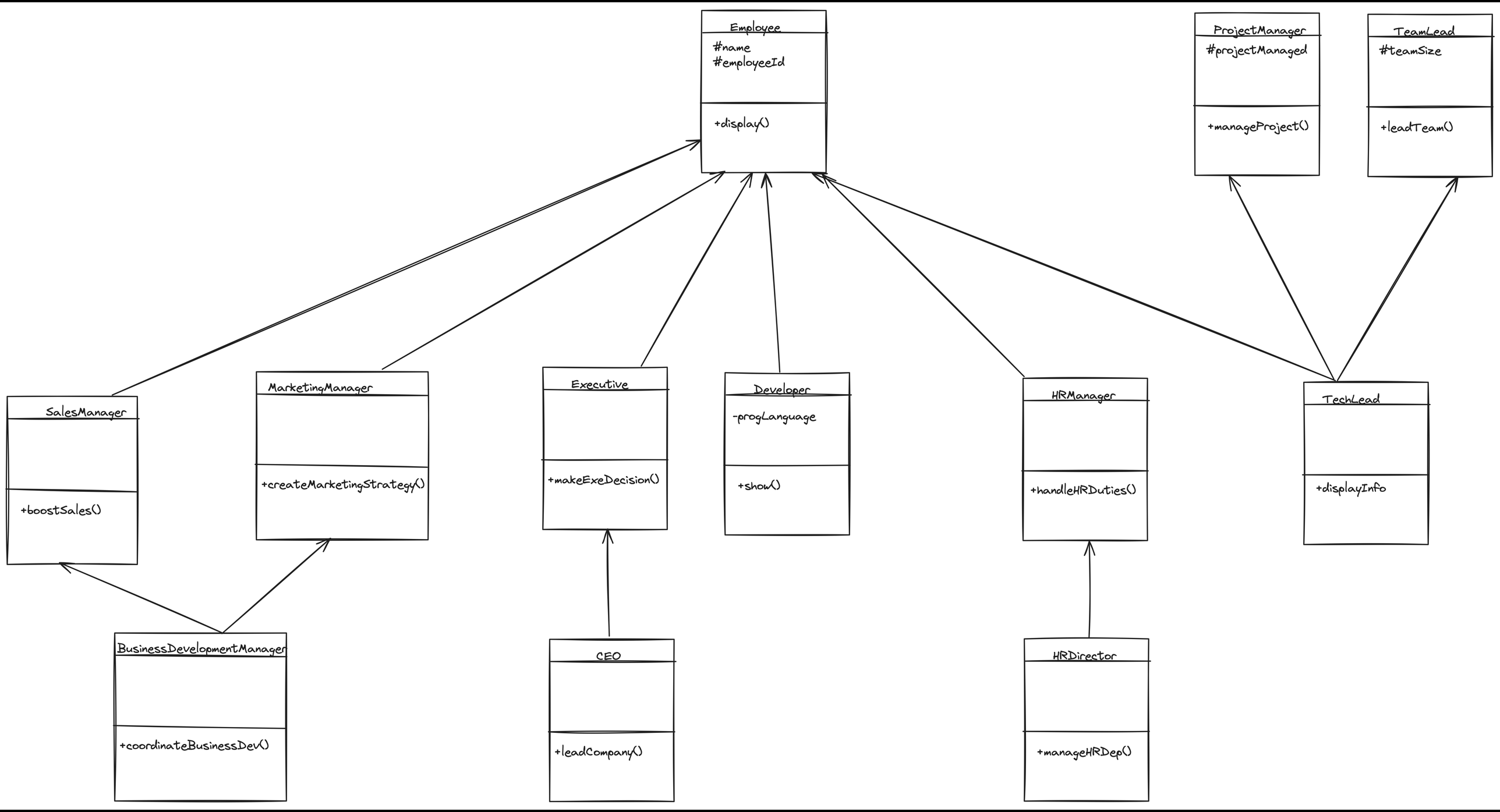
Protected Members

1. Members declared protected are accessible within class itself & to its derived class.
2. The protected access modifier serves two primary purposes:
    1. Encapsulation: Like the private access modifier, protected provides a level of encapsulation, ensuring that certain class members are not directly accessible from outside the class. This promotes data hiding and prevents external code from modifying or accessing sensitive data directly.
    2. Inheritance: Unlike private, protected members can be inherited by derived classes. This means that subclasses have limited access to these members, allowing them to build upon the base class's functionality while maintaining some level of data integrity and control.

Protected Members

Types of Inheritance




Employee
#name
#employeeId
+display()
ProjectManager
#projectManaged
+manageProject()
TeamLead
#teamSize
+leadTeam()
SalesManager
+boostSales()
MarketingManager
+createMarketingStrategy()
Executive
+makeExeDecision()
Developer
-progLanguage
+show()
HRManager
+handleHRDuties()
TechLead
+displayInfo
BusinessDevelopmentManager
+coordinateBusinessDev()
CEO
+leadCompany()
HRDirector
+manageHRDep()

Advantages

1. Reusability: Avoids duplicating methods in child classes that already exist in parent classes.

2. Code modification: Localises changes, preventing inconsistencies throughout the program.

3. Extensibility: Allows easy enhancement or upgrade of specific parts of a product without altering core attributes.

4. Data hiding: Supports encapsulation by keeping some data private in the base class, preventing alteration by derived classes.

Implementing Encapsulation in C++

Implementing Encapsulation in Java

1. Java does not support multiple inheritance.